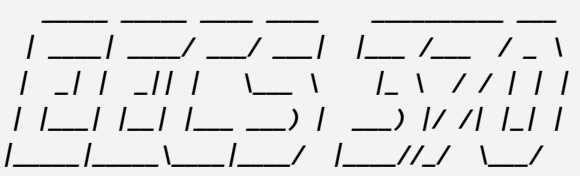
Final Exam



EECS 370 Winter 2022: Intro to Computer Organization

You are to abide by the University of Michigan College of Engineering Honor Code. Please sign below to signify that you have kept the honor code pledge:

I have neither given nor received aid on this exam,
nor have I concealed any violations of the Honor Code.

Signature: _____

Name: _____

Uniqname: _____

First/Last name of person sitting to your ***Right***
(Write ⊥ if you are at the end of the row) _____

First/Last name of person sitting to your ***Left***
(Write ⊥ if you are at the end of the row) _____

Exam Directions:

- You have **120 minutes** to complete the exam. There are **8** questions in the exam on **15** pages (double-sided). **Please flip through your exam to ensure you have all 15 pages.**
- You must show your work to be eligible for partial credit!
- Write legibly and dark enough for the scanners to read your answers.
- **Write your uniqname on the line provided at the top of each page.**

Exam Materials:

- You are allotted **one 8.5 x 11 double-sided** note sheet to bring into the exam room.
- You are allowed to use calculators that do not have an internet connection. All other electronic devices, such as cell phones or anything or calculators with an internet connection, are strictly forbidden and usage will result in an Honor Code violation.

1.	**Short Questions**	**_____ / 12 pts**
2.	**Branch Prediction**	**_____ / 11 pts**
3.	**LC2K Pipeline Datapath Performance**	**_____ / 10 pts**
4.	**The 3 C's of Caches**	**_____ / 9 pts**
5.	**New LC2K Pipeline Datapath**	**_____ / 15 pts**
6.	**Cache Locality**	**_____ / 15 pts**
7.	**VM Simulation**	**_____ / 16 pts**
8.	**VM Performance / Cache Performance**	**_____ / 12 pts**
	TOTAL	**_____ / 100 pts**

1.	**Short Questions**	**[12 pts]**
	Complete the following true/false and short answer questions	

True/False Questions [5 pts]

Circle One:

(a) Using the speculate-and-squash method to resolve control hazards can result in the same CPI as the detect-and-stall method. True / False

(b) Increasing the associativity of a cache can reduce capacity misses. True / False

(c) If a machine has infinite physical memory, virtual memory is no longer needed. True / False

(d) A multi-level page table can consume more space than a single-level page table. True / False

(e) Virtual address to physical address translation can happen simultaneously with the cache access. True / False

Short Answer Questions

(f) **[2 pts]** Consider a cache with a given size. You are asked to change the associativity and block size to reduce the tag area overhead:

i) How would you change the associativity? Circle around your choice.

Increase Associativity , Decrease Associativity

ii) How would you adjust the block size? Circle around your choice.

Increase Block Size , Decrease Block Size

(g) **[3 pts]** Consider the following assembly code being simulated on the **5-stage** LC2K pipeline datapath that uses **detect and forward** for data hazards, **speculate not-taken and squash** for control hazards, and **internal forwarding** for register file.

i) Circle around the registers of the executed instructions that are going to use forwarded-data from pipeline registers (including the internal forwarding of register file):

Example: `nor 5 6 7`

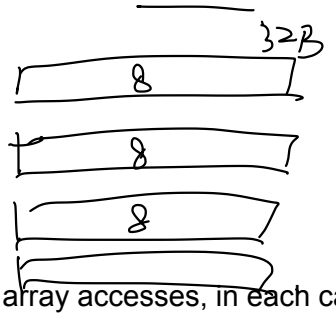
1	lw	0	1	10
2	lw	0	2	10
3	beq	1	2	0
4	add	1	2	3
5	nor	1	3	4
6	sw	3	4	10
7	halt			

ii) How many cycles would it take to complete the program?

Answer: 15

(h) **[2 pts]** Consider a byte-addressable system with a 128 B fully-associative cache that has a block size of 32 B. The following lines of code are executed:

```
1  int data[20];
2  for(unsigned int i = 0; i < 20; i++) {
3        data[i] = data[i] + i;
4  }
```



How many bytes would be written into main memory for `data` array accesses, in each case of write-through and write-back policy for the cache. You can assume `data` starts from address 0 and all dirty blocks are written *in their entirety* back to cache at the end of the program.

Write-through: 80 B

Write-back: 96 B

2.	**Branch Prediction**	**[11 pts]**
	Simulate an LC2K assembly program and answer questions about its branches	

Consider the following LC2K assembly program running on our 5-stage pipelined datapath from lecture. This program **counts the number of zeros** in the binary array **Arr** and stores that count in **r5**. Assume all registers are initialized to zero. Answer the following question about the program's branch decisions.

```
1               lw      0   1   One     //r1 = 1
2               lw      0   2   Len     //r2 = Len
3               lw      0   3   Zero    //r3 = loop counter i
4   loop        beq     3   2   end     //loop terminates if i==Len
5               lw      3   4   Arr     //load Arr[i]
6               beq     4   1   cont    //if Arr[i]==1, skip Line 7
7               add     5   1   5       //increment r5
8   cont        add     3   1   3       //increment i
9               beq     0   0   loop
10  end         halt
11  Zero        .fill   0
12  One         .fill   1
13  Len         .fill   3
14  Arr         .fill   1
15              .fill   1
16              .fill   0
```

(a) [7 pts] Write the sequence of branch decisions for each **beq** instruction, using **T** to represent "taken" and **N** to represent "not taken." (You might not need all the boxes.)

Line 4 **beq** (Loop termination condition) N N N T

Line 6 **beq** (If-Condition) T T N

Line 9 **beq** (Loop condition) T T T

Combined sequence, as seen globally N T T N T T N N T T

(b) [4 pts] What percentage of the total branch decisions are correctly predicted when using the following prediction schemes? (You can leave your answers as fractions.)

Predict always not taken (N) 4/10

Predict backwards taken (T), forwards not taken (N) 7/10

Predict using a local (*i.e.*, per **beq** instruction) 2-bit branch predictor intialized to the starting state "strongly taken" 6/10

3.	**LC2K Pipeline Datapath Performance**	**[10 pts]**
	Calculate the CPI and execution time for a pipelined datapath	

Consider an LC2K assembly program running on our 5-stage pipelined datapath from lecture, which has internal forwarding for the register file. The program has the following instruction breakdown:

- R-type instructions (**`add`** and **`nor`**) 50%
- Load instructions (**`lw`**) 25%
- Store instructions (**`sw`**) 15%
- Branch instructions (**`beq`**) 15%

In the program, 60% of branches are not taken (N). Additionally, 20% of the R-type instructions are followed immediately by a dependent R-type instruction (e.g., **`add 1 1 2`**, **`add 2 2 3`**) and 80% of load instructions are followed immediately by a dependent R-type instruction (e.g., **`lw 0 3 0`**, **`add 3 3 4`**). There are no other data dependencies.

(a) **[4 pts]** What is the CPI of the program when the processor uses **detect-and-stall** for data hazards and **speculate-and-squash** for control hazards with a branch predictor that always predicts not taken (N). Please show your work for partial credit.

2

CPI = 1 + 0.5 × 0.2 × 2 + 0.25 × 0.8 × 1 + 0.15 × 0.4 × 3

(b) **[4 pts]** What is the CPI of the program when the processor uses **detect-and-forward** for data hazards and **speculate-and-squash** for control hazards with a branch predictor that always predicts not taken (N). Please show your work for partial credit.

CPI = 1 + 0.25 × 0.8 × 1 + 0.15 × 0.4 × 3

(c) **[2 pts]** Given that the processor frequency is 1MHz and the program executes one million instructions, how many seconds faster is the execution time of the program when using detect-and-forward? Show your work by calculating the execution time in seconds for parts **(a)** and **(b)**, then finding the difference between the two.

1.14 - 1.18 = 0.4

$\frac{1}{1\text{MHz}} = \frac{1}{1\times10^6} = 1000\text{ns}$.

0.4 × 1 × 10^6 × 1000ns.

1 × 10^9 × 0.4 ns = 4 × 10^8 ns. = 0.4s

4. The 3 C's of Caches [9 pts]

Determine if the memory references result in compulsory, capacity, or conflict misses

Consider an 8-bit processor with byte-addressable memory that has a cache with the following configuration:

- **Cache size** 32 B
- **Block size** 16 B
- **Associativity** Direct mapped (1-way)
- **Replacement Policy** Least-recently used (LRU)

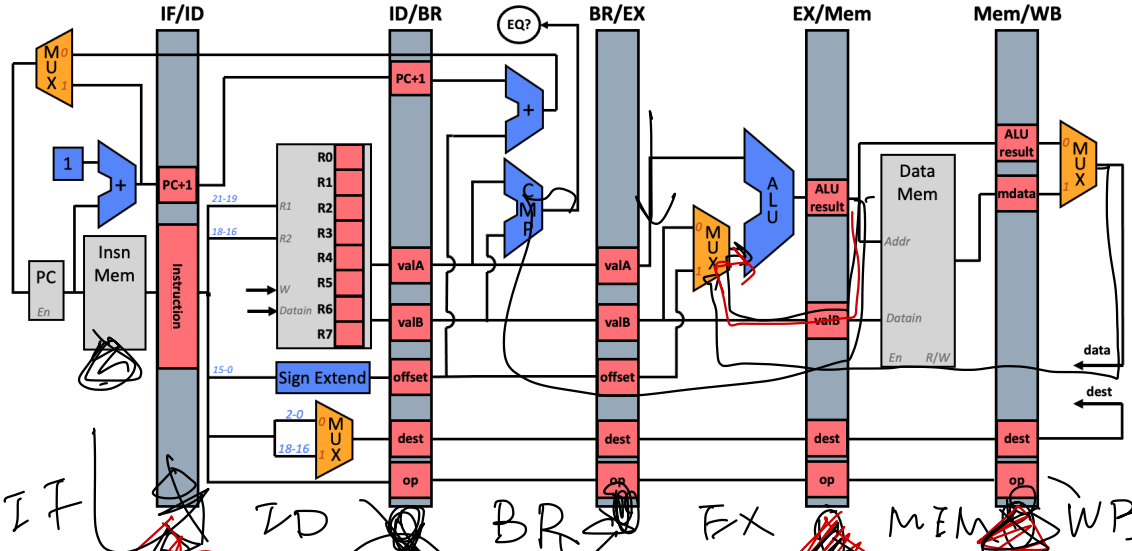
Assume the cache is initially empty. Fill out the table given the provided sequence of memory references. If the access was a miss, fill in the bubble (○) to indicate the type of miss. If a block was evicted because of the access, write the tag of the block that was evicted.

Reference Address in Binary (0b)	Tag	Set Index	Hit or Miss?	Type of Miss (if applicable)	Tag of evicted block (if applicable)
0b1001 0110	0b 100	0b 1	M	● Compulsory ○ Capacity ○ Conflict	0b
0b1110 1111	0b 111	0b 0	M	● Compulsory ○ Capacity ○ Conflict	0b
0b1001 0010	0b 100	0b 1	H	○ Compulsory ○ Capacity ○ Conflict	0b
0b0110 1011	0b 011	0b 0	M	● Compulsory ○ Capacity ○ Conflict	0b 111
0b1110 0000	0b 111	0b 0	M	○ Compulsory ● Capacity ○ Conflict	0b 011
0b0110 1000	0b 011	0b 0	M	○ Compulsory ○ Capacity ● Conflict	0b 111

5. New LC2K Pipeline Datapath [15 pts]

Identify the hazards and pipeline stalls in the new LC2K pipelined-datapath

Consider the following revised LC2K pipelined-datapath, in which a new pipeline stage called ***B**ranch **R**esolve* (***BR***) is added between *ID* and *EX* stage, so we can resolve control hazards earlier in the pipeline. The new *BR* stage includes an adder to calculate the branch target as well as a comparator to check the equality of regA and regB values.



(a) [6 pts] Assume that this new pipeline design has **only the original data forwarding paths** of the 5-stage LC2K pipeline (i.e., *EX/MEM→EX*, *MEM/WB→EX*, and internal forwarding for the register file). Compute how many cycles we will need to stall the pipeline for each of the following scenarios of instructions containing data dependencies.

In the table, "Distance" denotes the distance between dependent instructions. When Distance = 0, the instructions are adjacent (e.g., `add 1 1 2`, `nor 1 2 3`). When the Distance = 1, there is one instruction separating the dependent instructions (eg., `add 1 1 2`, <unrelated instruction>, `nor 1 2 3`), and so on.

Source of Data Dependency	Dependent Instruction	# of stalled cycles		
		Distance = 0	Distance = 1	Distance = 2
`R-type (add 1 1 2)`	`R/I-type (nor 1 2 3)`	0 cycles	0 cycles	0 cycles
`R-type (add 1 1 2)`	`Branch (beq 1 2 end)`	2	1	0
`Load (lw 0 2 0)`	`R/I-type (nor 1 2 3)`	1	0	0
`Load (lw 0 2 0)`	`Branch (beq 1 2 end)`	3	2	1

(b) [3 pts] If we allow new data-forwarding paths, what is the **minimum set** of forwarding paths that are necessary to **minimize** the number of stalling cycles for part (a)? (You might not need all the paths.)

	Source Pipeline Register		Destination Stage
Example:	*EX/MEM*	→	*EX*
New Forwarding Path 1:	EX/MEM	→	BR
New Forwarding Path 2:	MEM/WB	→	BR
New Forwarding Path 3:		→	

(c) [5 pts] With the original data forwarding paths of the 5-stage LC2K pipeline and the **new data forwarding paths** from part (b), compute how many cycles we will need to stall the pipeline for the following scenarios of instructions containing data dependencies.

Source of Data Dependency	Dependent Instruction	# of stalled cycles		
		Distance = 0	Distance = 1	Distance = 2
`R-type (add 1 1 2)`	`R/I-type (nor 1 2 3)`	0 cycles	0 cycles	0 cycles
`R-type (add 1 1 2)`	`Branch (beq 1 2 end)`	1	0	0
`Load (lw 0 2 0)`	`R/I-type (nor 1 2 3)`	1	0	0
`Load (lw 0 2 0)`	`Branch (beq 1 2 end)`	2	1	0

(d) [1 pts] With the new pipeline design, how many cycles of stalls are we going to have in the pipeline in case of a branch misprediction?

Stalled Cycles: 2

6.	**Cache Locality**	**[15 pts]**
	Trace memory accesses and determine the ideal cache layout for a C++ program	

Consider the following convolutional kernel executing on a system with a byte-addressable memory. Load/store instructions operate with 4 B granularity (i.e., each load/store instruction returns a 4 B value from the cache to the processor, resulting in a *single* cache hit or miss).

```
1   #define N 3
2   int Out[N], Input[N], Mask[3];
3
4   //initialization code
5
6   for(int i=0; i<N; i++)
7   {
8      dot=0;
9      for(int j=0; j<3; j++)
10     {
11        if((i+j-1)>=0 && (i+j-1)<N)
12           dot += Input[i+j-1] * Mask[j];
13     }
14     Out[i] = dot;
15  }
```

For this program:

- All variables except the arrays **`Out`**, **`Input`**, and **`Mask`** are mapped to registers. Hence, only these arrays require loads/stores to the cache.
- **`Input`** starts at address `0x100`, **`Mask`** starts at address `0x200`, and **`Output`** starts at address `0x400`.
- On line 14, the **`Input`** array is accessed before the **`Mask`** array.

(a) [2.5 pts] Write down the memory addresses for the array accesses in the program. For example, the iteration i=0 accesses `Input[0]`, `Mask[1]`, `Input[1]`, `Mask[2]`, `Out[0]` and the memory addresses for this iteration are completed in the table below.

for i=0	**0x100, 0x204, 0x104, 0x208, 0x400**
for i=1	**0x100, 0x200, 0x104, 0x204, 0x108, 0x208, 0x404**
for i=2	0x104 0x200 0x108 0x204 0x408

Total number of memory accesses: 17

0x100 0x10 0 miss
0x204 0x20 0 miss
0x104 0x10 0 hit
0x208 0x20 1 miss
0x400 0x40 0 miss
0x100 0x10 0 miss
0x200 0x20 0 miss
0x104 0x10 0 hit
0x204 0x20 0 hit
0x108 0x10 1 miss
0x208 0x20 1 miss

(b) [3 pts] Say our processor has a fully associative cache size of 16 B and block size of 8 B with LRU replacement policy. Determine hits/misses for the above program. Assume the memory address has 12 bits. Show your work.

0x404 0x40 0 miss

tag | 0x100 0x400 0x200 0x201 0x100 0x104 0x401

tag | 0x200 0x201 0x100 0x101 0x400 0x200

0x104 0x10 0 miss 0x108 0x10 1 miss 0x408 0x40 1 miss
0x200 0x20 0 miss 0x204 0x20 0 hit

# of Hits: 4

# of Misses: 13

i=0 M M H M M
i=1 M M H H M M M
i=2 M M M H M

(c) [4.5 pts] What is the minimum size of a fully associative cache with the same block size as above (8 B) that can eliminate all misses except compulsory misses. Determine hits/misses for this new cache. **Note that cache size does not need to be a power of 2**. Show your work.

i=0

0x100	0x10	0	miss
0x204	0x20	0	miss
0x104	0x10	0	hit
0x208	0x20	1	miss
0x400	0x40	0	miss

i=1

0x100	0x10	0 hit
0x200	0x20	0 hit
0x104	0x10	0 hit
0x204	0x20	0 hit
0x108	0x10	1 miss

i=2

0x104	0x10	0 hit
0x200	0x20	0 hit
0x108	0x10	1 hit
0x204	0x20	1 hit
0x408	0x40	1 miss

0x10 0
0x20 0
0x20 1
0x40 0
0x10 1

Size = 5.

Cache Size (B): 5x8 = 40 Bytes

0x208 0x20 1 hit.

# of Hits: 11

0x404 0x40 0 hit.

# of Misses: 6

i=0 M M H M M
i=1 H H H H M H H
i=2 H H H H M

(d) [5 pts] Say you can increase the cache size to 64 B. What cache configuration for this increased size will result in the lowest number of cache misses? Determine hits/misses for your new cache configuration. Show your work.

0x100 0x104 0x108 0x200 0x204 0x208. 0x400 0x404 0x408.

12 $2^4 \Rightarrow 16$

4x16 = 64.

i=0

0x100	0x10	miss
0x204	0x20	miss
0x104	0x10	hit
0x208	0x20	hit
0x400	0x40	miss

0x100	0x10	hit
0x200	0x20	hit
0x104	0x10	hit
0x204	0x20	hit
0x108	0x10	hit
0x208	0x20	hit
0x404	0x40	hit

0x104 0x10 hit
0x200 0x20 hit
0x108 0x10 hit
0x204 0x20 hit
0x408 0x40 hit

0x10
0x20
0x40

Associativity: full-association.

Block Size (B): 16 Bytes

# of Hits: 14

# of Misses: 3

7.	**Virtual Memory and Page Table**	**[16 pts]**
	Simulate virtual to physical page translation and calculate page table overheads	

Consider a 32-bit processor that has a byte-addressable memory with the following configuration:

- **Page Size** 256 B
- **Virtual Memory Size** 16 KB
- **Physical Memory Size** 2 KB (8 pages)
- **Page Replacement Policy** Least-recently used (LRU)
- **Page Table Layout** Single-level page table
- **Page Table Size** 256 B
- **Page Table Entry Size** 4 B

Physical page 0x0 is reserved for the operating system (OS) and cannot be replaced. Similarly, physical pages 0x1 and 0x2 are reserved for the current processes' page tables and cannot be replaced. On a page fault, the page table is updated before allocating a physical page. If more than one free page is available, the smallest physical page number is chosen.

Assume that two processes with Process ID (PID) 370 and 281, respectively, have been running. The initial state of physical memory is shown below:

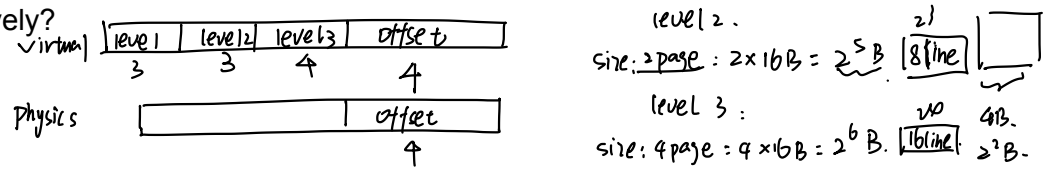
Physical Page # (PPN)	Memory Contents
0x0	Reserved for OS
0x1	PID 370 Page Table
0x2	PID 281 Page Table
0x3	PID 370: VPN 0x3
0x4	PID 370: VPN 0xA
0x5	PID 281: VPN 0x2
0x6	PID 281: VPN 0x3
0x7	PID 370: VPN 0xF

(a) [10 pts] Complete the following table for the given sequence of virtual address requests. Please express your solution in hexadecimal.

Time	PID	Virtual Address (VA)	Virtual Page # (VPN)	Physical Page # (PPN)	Page Fault? (Y/N)	Physical Address (PA)
0	370	0x31A	0x 3	0x 3	N	0x 31A
1	370	0xA02	0x A	0x 4	Y	0x 402
2	281	0x31A	0x 3	0x 6	Y	0x 61A
3	370	0xF00	0x F	0x 7	N	0x 700
4	281	0x618	0x 6	0x 5	Y	0x 518

(b) Suppose the page size is reduced to **16 B** and the single-level page table is replaced by a **3-level page table**, in which the size of each 2nd level page table is **2 pages** and the size of each 3rd level page table is **4 pages**. Answer the following questions about this new configuration. (Note that the page table entry size is still 4 B)

i) [2 pts] How many index bits are required for the 2nd level and the 3rd level page table, respectively?



2nd level: 3 index bits

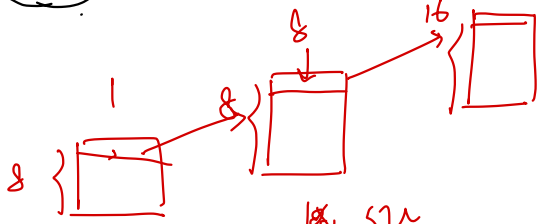
3rd level: 4 index bits

14 - 4 - 4 - 3 = 3.

ii) [2 pts] What is the **minimum** storage space this 3-level page table would occupy in bytes?

min → only have level 1

2^3 = 8 line. 8 x 4B = 32B.



iii) [2 pts] What is the **maximum** storage space this 3-level page table would occupy in bytes?

max → All full.

level 2: $2^3 \times 2^3 \times 4 = 2^8$

level 3: $2^3 \times 2^3 \times 2^6 \times 4 = 2^{12}$ 32B + 2^{11}B + 2^{16}B.

level 1 : 32B

level 2 : $2^3 \times 2^3 = 2^6$ ~~$2^6 \times 2^5 = 2^{11}$ B.~~

level 3 : $2^3 \times 2^3 \times 2^4 \times 2^6$ B ~~$2^6 \times 2^{10}$ B~~ $= 2^{16}$ B.

8.	**VM + Cache Performance**	**[12 pts]**
	Optimize the virtual memory system based on latency-cost tradeoff	

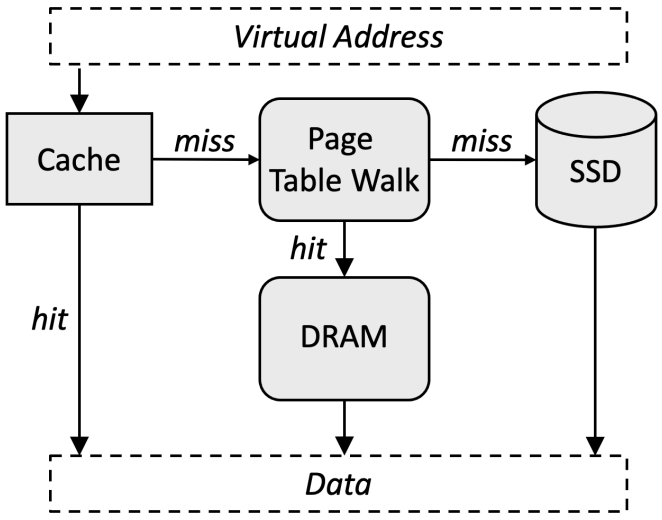
You have been tasked to configure the memory system for a custom processor designed by a startup company. The company has placed the following limitations on the design:

- The cache should be **virtually addressed** (virtual to physical address translation occurs in case of the cache miss)
- Virtual memory should use a single-level page table
- There is no TLB in the memory system.

The company is considering the following technologies the memory system components:

Component	Hit Rate	Technology	Latency	Cost
Cache	80%	SRAM-A	2 ns	$20
		SRAM-B	1 ns	$40
DRAM (Main Memory)	90%	DDR3	20 ns	$15
		DDR4	10 ns	$45
SSD (Disk)	100%	SATA 3.0	800 ns	$20

(a) **[6 pts]** Considering this memory system, write down the coefficients for the average memory access time (AMAT) equation. The variables LC, LD, and LS represent the latencies for the cache, DRAM, and SSD components, respectively.
You can assume that any necessary updates to cache, page table, and DRAM due to data miss would happen in parallel to data retrieval (No impact on AMAT). Please show your work.



$LC + 0.2 \times (LD)$

$+ 0.2 \times 0.9 \times LD$

$+ 0.2 \times 0.1 \times LS$

0.38

$AMAT = \alpha \times LC + \beta \times LD + \gamma \times LS$

$\alpha =$ 1 , $\beta =$ 0.2 , $\gamma =$ 0.2 x 0.1

$LC + 0.2LD + 0.18LD + 0.02LS$

$LC + 0.38LD + 0.02LS$

(b) [4 pts] The company has set the following restrictions for the memory system:

- Average memory access time (AMAT) must be **less than 25 ns.**
- The memory system components must **cost less than $100 in total**.

Circle the ideal technologies for the cache and DRAM to **minimize the latency while meeting the cost budget**. Please show your work.

Cache: SRAM-A , SRAM-B

DRAM: DDR3 , DDR4

SSD: **SATA 3.0**

$1 \times 1 + 0.2 \times 20 = 1 + 4 = 5$

$1 \times 2 + 0.2 \times 10 = 2 + 2 = 4$

$4 + 0.02 \times 80$

$4 + 16 = 20$

$20 + 65 + 20$

85

(c) [2 pts] Using the equation in part (a), what would the AMAT and total cost be for the technologies selected in part (b)?

AMAT = ~~20~~ 21.8 , Cost = ~~85~~

$1 \times 2 + 0.38 \times 10 + 0.02 \times 80$

$= 2 + 3.8 + 16$ $18 + 3.8 = 21.8$